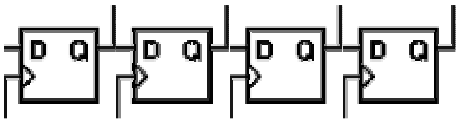
Section 14

Shift Registers

Finite State Machines (FSM)

Shift Registers

- Shift Registers are a useful part of digital design in:
  - Serial to Parallel Conversion.
  - Pattern Detection
  - Element Extraction
  - Etc.



Shift Registers – Serial to Parallel

- The data input stream is scalar (1-bit) the data output stream is an 8-bit vector.
- The data output clock will be 1/8th the speed of the input clock.

```
module sertopar (DOUT,DIN,INCLK,OUTCLK,RESET);
  input DIN, INCLK, OUTCLK, RESET;
  output [7:0] DOUT;
  reg [7:0] DOUT, shftreg;

  always @(posedge INCLK or posedge RESET) begin
     if (RESET) shftreg <= 0;
    else begin
       shftreg[7] <= shftreg[6];
       shftreg[6] <= shftreg[5];
       shftreg[5] <= shftreg[4];
       shftreg[4] <= shftreg[3];
       shftreg[3] <= shftreg[2];
       shftreg[2] <= shftreg[1];
       shftreg[1] <= shftreg[0];
       shftreg[0] <= DIN;
    end
  end

  always @(posedge OUTCLK or posedge RESET) begin
    if (RESET) DOUT <= 0;
    else  DOUT <= shftreg;
  end
endmodule
```

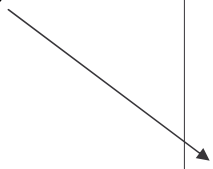
Shift Registers – Serial to Parallel

- Alternate Form for Shift Register.
  - Move blocks when shifting rather than bits.

```
module sertopar (DOUT,DIN,INCLK,OUTCLK,RESET);

  input DIN, INCLK, OUTCLK, RESET;
  output [7:0] DOUT;

  reg [7:0] DOUT, shftreg;

  always @(posedge INCLK or posedge RESET) begin
    if (RESET)  shftreg <= 0;
    else begin
       shftreg[7:1] <= shftreg[6:0];
       shftreg[0] <= DIN;
    end
  end
  always @(posedge OUTCLK or posedge RESET) begin
    if (RESET) DOUT <= 0;
    else DOUT <= shftreg;
  end

endmodule
```

Shift Register - Serial to Parallel

- Testbench
  - Why is OUTCLK delayed by 1ms?



```
`timescale 1ms/10us
module tb;

  reg DIN, INCLK, OUTCLK, RESET;
  integer i;
  wire [7:0] DOUT;

  sertopar u1 (DOUT,DIN,INCLK,OUTCLK,RESET);

  always #10 INCLK = ~INCLK;

  initial begin
    #1;
    forever #80 OUTCLK = ~OUTCLK;
  end

  initial begin
    {DIN, INCLK, OUTCLK, RESET} = 0;
    #1 RESET = 1;
    #2 RESET = 0;
    for (i = 0; i < 2048; i = i + 1) begin
        @(negedge INCLK) DIN = $random;
    end
    #100 $finish;
  end
endmodule
```

Shift Registers – Pattern Detection

- PATTERNMATCH will go high whenever the value of the shift register matches the PATTERN parameter.
  - Use conditional constructs to find patterns in Data.

```
module pattern (PATTERNMATCH,DIN,CLK,RESET);

  input DIN, CLK, RESET;
  output PATTERNMATCH;

  parameter PATTERN = 8'b11001100;

  reg [7:0] shftreg;

  assign PATTERNMATCH = (PATTERN == shftreg) ? 1 : 0;

  always @(posedge CLK or posedge RESET) begin
    if (RESET) begin
       shftreg <= 0;
    end
    else begin
       shftreg[7:1] <= shftreg[6:0];
       shftreg[0] <= DIN;
    end
  end
endmodule
```

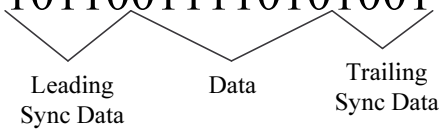
Shift Registers – Pattern Detection

- Testbench

```
`timescale 1ms/10us
module tb;
  reg DIN, CLK, RESET;
  integer i;

  pattern u1 (PATTERNMATCH,DIN,CLK,RESET);

  always #10 CLK = ~CLK;

  initial begin
    {DIN, CLK, RESET} = 0;
    #1 RESET = 1;
    #2 RESET = 0;
    repeat (2) begin
        for (i = 0; i < 64; i = i + 1) begin
          @(negedge CLK) DIN = $random;
        end
        #100;
        @(negedge CLK) DIN = 1;
        @(negedge CLK);
        @(negedge CLK) DIN = 0;
        @(negedge CLK);
        @(negedge CLK) DIN = 1;
        @(negedge CLK);
        @(negedge CLK) DIN = 0;
        @(negedge CLK);
    end
    #100 $finish;
  end
endmodule
```

Shift Registers – Element Extraction

- Data can be embedded in synchronization code.
- For Example 8 bit data embedded in 5 sync bits of leading data, and 4 sync bits of trailing data.
  - Data stream :



Leading Sync Data | Data | Trailing Sync Data

```
module elementextract (DOUT,DIN,CLK,RESET);

  input DIN, CLK, RESET;
  output [7:0] DOUT;

  reg [16:0]        shftreg;
  reg [7:0]         DOUT;

  parameter  leadingsync = 5'b10110;
  parameter  trailingsync = 4'b1001;

  always @(posedge CLK or posedge RESET) begin
    if (RESET) begin
        DOUT <= 0;
        shftreg <= 0;
    end
    else begin
        shftreg[16:1] <= shftreg [15:0];
        shftreg[0] <= DIN;
        if ((shftreg[16:12] == leadingsync) & (shftreg[3:0] == trailingsync))
          DOUT = shftreg[11:4];
        else
          DOUT = 'bz;
    end
  end

endmodule
```

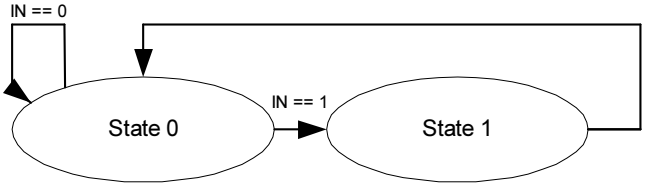
Shift Registers – Element Extraction

- Testbench

```
`timescale 1ms/10us
module tb;

  reg CLK, RESET, DIN;
  integer i;
  wire [7:0] DOUT;

//  leadingsync = 5'b10110;  trailingsync = 4'b1001;

  elementextract u1 (DOUT,DIN,CLK,RESET);

  always #10 CLK = ~CLK;

  initial begin
    {RESET,CLK} = 0;
    #2 RESET = 1;
    #1 RESET = 0;
    repeat (40) @(negedge CLK) DIN = $random;
    repeat (10) begin
          @(negedge CLK) DIN = 1;
          @(negedge CLK) DIN = 0;
          @(negedge CLK) DIN = 1;
          @(negedge CLK);
          @(negedge CLK) DIN = 0;
          repeat (8) @(negedge CLK) DIN = $random;
          @(negedge CLK) DIN = 1;
          @(negedge CLK) DIN = 0;
          @(negedge CLK);
          @(negedge CLK) DIN = 1;
    end // repeat (10)
    #100 $stop;
  end

endmodule
```

Finite State Machines

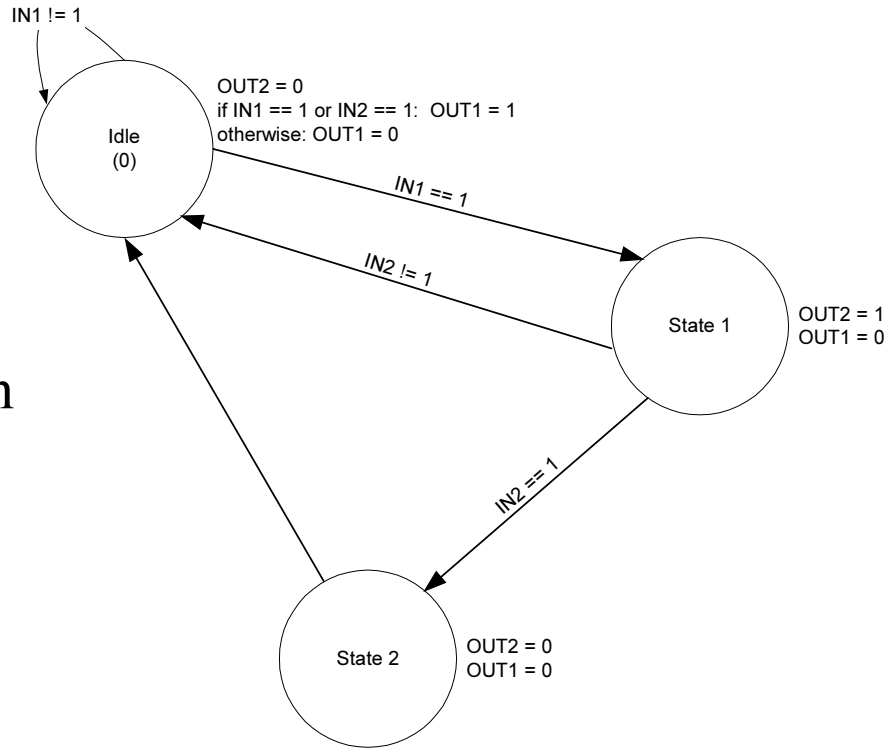
- Finite State Machine Elements:
  - A state register is defined
  - States defined as Parameters.
    - Not necessary, but much easier to read and program.
  - State changes handled in case statements.




```
module explicit_fsm (OUT, CLK, IN, RESET);
  input CLK, IN, RESET
  output OUT;
  reg STATE, OUT;
  parameter state0 = 0;
  parameter state1 = 1;
always @(posedge CLK or posedge RESET)
  if (RESET)
    {STATE,OUT} <= 2'b00;
  else
    case (STATE)
      state0:  begin
                 OUT <= 0;
                 if (~IN)  STATE <= 0;
                 else STATE <= 1;
               end
      state1:  begin
                 OUT <= IN;
                 STATE <= 0;
               end
      default: {STATE,OUT} <= 2'b00;
  endcase
endmodule
```

Finite State Machines – Example 1

- State Diagram



Finite State Machines – Example 1

RTL Code

```
module fsm (OUT1,OUT2,CLK,IN1,IN2,RESET);

  input CLK, RESET, IN1, IN2;
  output OUT1,OUT2;
  reg OUT1, OUT2;
  reg [1:0] current_state, next_state;

  parameter idle = 0;
  parameter state1 = 1;
  parameter state2 = 2;

always @ (posedge CLK or posedge RESET)
  if (RESET)
    current_state <= idle;
  else
    current_state <= next_state;

always @ (current_state or IN1 or IN2)begin

  case(current_state)
```

```
idle : begin
      OUT2 = 0;
      if (IN1 || IN2)  OUT1 = 1;
      else OUT1 = 0;
      if(IN1 == 1)  next_state = state1;
      else  next_state = idle;
   end
 state1 : begin
    OUT2 = 1;  OUT1 = 0;
    if(IN2 == 1)  next_state = state2;
    else next_state = idle;
 end
 state2 : begin
    OUT2 = 0;  OUT1 = 0;
    next_state = idle;
 end
 default : begin
    OUT1 = 1'b1;    OUT2 = 1'b1;
    next_state = idle;
 end
 endcase
end

endmodule
```


Some designers will explicitly encode their state variables to avoid glitches (i.e. Gray code or One-Hot). Others will add synthesis constraints to force encoding, such as:

#Synopsys Design Compiler FSM Encoding Constraint

set_fsm_encoding_style one_hot

At synthesis, this would change the state encoding to:

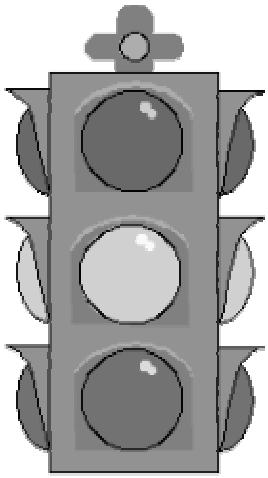
idle = 3'b001;

state1 = 3'b010;

state2 = 3'b100;

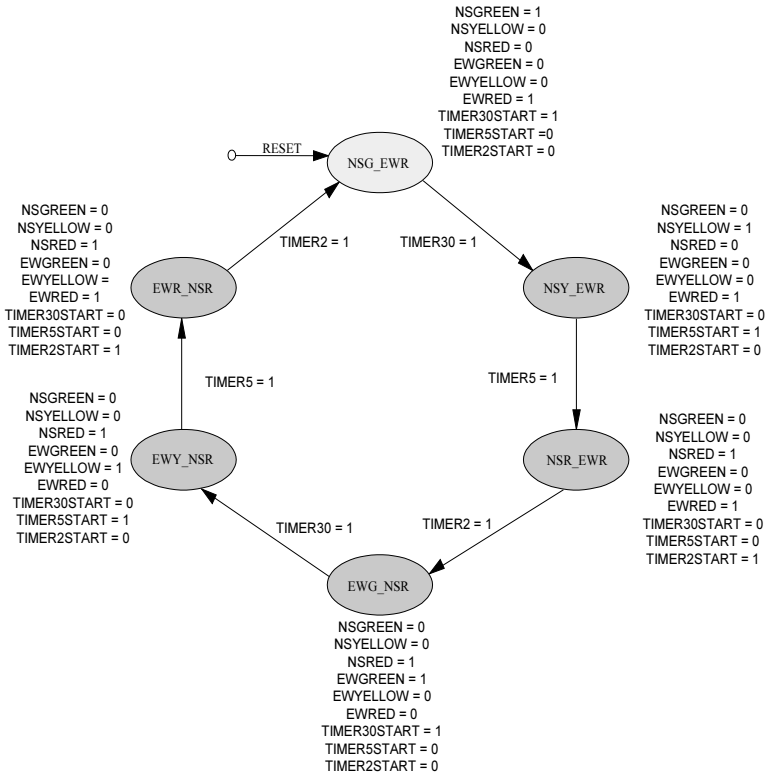
Finite State Machine – Example 2

- Traffic Light Controller
  - Simple intersection light with East-West and North-South control.



Finite State Machine – Example 2

- Default (NSG_EWR) is North-South light green, East-West light red.
- Light will change from green to yellow after 30 seconds.
- Light will remain yellow for 5 seconds.
- Both lights will be red for two seconds.
- External timers will be controlled by the State Machine.



Finite State Machine – Example 2

```
module trafficlight (NSGREEN, NSYELLOW, NSRED, EWGREEN,
                     EWYELLOW, EWRED, TIMER30START,
                     TIMER5START, TIMER2START, CLK, RESET,
                     TIMER30, TIMER5, TIMER2);

  input  CLK, RESET, TIMER30, TIMER5, TIMER2;
  output NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW,
         EWRED, TIMER30START, TIMER5START, TIMER2START;

  reg NSGREEN, NSYELLOW, NSRED, EWGREEN, EWYELLOW,
      EWRED,  TIMER30START, TIMER5START, TIMER2START;

  reg [2:0] STATE, NEXTSTATE;

  parameter NSG_EWR = 0;
  parameter NSY_EWR = 1;
  parameter NSR_EWR = 2;
  parameter EWG_NSR = 3;
  parameter EWY_NSR = 4;
  parameter EWR_NSR = 5;

  always @(posedge CLK or posedge RESET) begin
    if (RESET) STATE <= state0;
    else STATE <= NEXTSTATE;
  end
```

```
always @(TIMER30, TIMER5, TIMER2, STATE) begin
    case (STATE)
      NSG_EWR : begin
        NSGREEN = 1;
        NSYELLOW = 0;
        NSRED = 0;
        EWGREEN = 0;
        EWYELLOW = 0;
        EWRED = 1;
        TIMER30START = 1;
        TIMER5START =0;
        TIMER2START = 0;
         if (TIMER30 == 1) NEXTSTATE = NSY_EWR;
         else NEXTSTATE = STATE;
      end
      NSY_EWR : begin
        NSGREEN = 0;
        NSYELLOW = 1;
        NSRED = 0;
        EWGREEN = 0;
        EWYELLOW = 0;
        EWRED = 1;
        TIMER30START = 0;
        TIMER5START = 1;
        TIMER2START = 0;
         if (TIMER5 == 1) NEXTSTATE = NSR_EWR;
         else NEXTSTATE = STATE;
       end
       NSR_EWR : begin …
```

Finite State Machine – Example 2

```
NSR_EWR : begin
  NSGREEN = 0;
  NSYELLOW = 0;
  NSRED = 1;
  EWGREEN = 0;
  EWYELLOW = 0;
  EWRED = 1;
  TIMER30START = 0;
  TIMER5START = 0;
  TIMER2START = 1;
  if (TIMER2 == 1) NEXTSTATE = EWG_NSR;
  else NEXTSTATE = STATE;  end
EWG_NSR : begin
  NSGREEN = 0;
  NSYELLOW = 0;
  NSRED = 1;
  EWGREEN = 1;
  EWYELLOW = 0;
  EWRED = 0;
  TIMER30START = 1;
  TIMER5START = 0;
  TIMER2START = 0;
  if (TIMER30 == 1) NEXTSTATE = EWY_NSR;
  else NEXTSTATE = STATE;   end
EWY_NSR : begin
  NSGREEN = 0;
  NSYELLOW = 0;
  NSRED = 1;
  EWGREEN = 0;
  EWYELLOW = 1;
  EWRED = 0;
  TIMER30START = 0;
  TIMER5START = 1;
  TIMER2START = 0;
  if (TIMER5 == 1) NEXTSTATE = EWR_NSR;
  else NEXTSTATE = STATE;  end
```

```
EWR_NSR : begin
   NSGREEN = 0;
   NSYELLOW = 0;
   NSRED = 1;
   EWGREEN = 0;
   NWYELLOW = 0;
   EWRED = 1;
   TIMER30START = 0;
   TIMER5START = 0;
   TIMER2START = 1;
   if (TIMER2 == 1) NEXTSTATE = NSG_EWR;
   else NEXTSTATE = STATE;  end
default : begin
  NSGREEN = 1;
  NSYELLOW = 0;
  NSRED = 0;
  EWGREEN = 0;
  EWYELLOW = 0;
  EWRED = 1;
  TIMER30START = 1;
  TIMER5START =0;
  TIMER2START = 0;
  NEXTSTATE = NSG_EWR;
end
endcase
end
endmodule
```

Finite State Machine – Example 2

- Synthesis tool should recognize the state machine.

```
=========================================
*            HDL Synthesis              *
=========================================

Synthesizing Unit <trafficlight>.
Related source file is trafficlight.v.
Found finite state machine <FSM_0>
for signal <STATE>.
-----------------------------------
| States             | 6
|
| Transitions        | 12
|
| Inputs             | 3
|
| Outputs            | 9
|
| Reset type         | asynchronous
|
| Encoding           | automatic
|
| State register     | d  flip-flops
|
-----------------------------------
Summary:
inferred 1 Finite State Machine(s).
Unit <trafficlight> synthesized.
=========================================
```

Review

- What uses to Shift Registers have?
- How do you shift data through a Shift Register?
- Are FSMs synthesizable?
- What is the recommended was to declare your state variables?